The Ability-Based Design Mobile Toolkit (ABD-MT): Developer Support for Runtime Interface Adaptation Based on Users' Abilities

JUNHAN KONG, The Information School, DUB Group, University of Washington, USA
MINGYUAN ZHONG, Paul G. Allen School of Computer Science & Engineering, DUB Group, University of Washington, USA
JAMES FOGARTY, Paul G. Allen School of Computer Science & Engineering, DUB Group, University of Washington, USA
JACOB O. WOBBROCK, The Information School, DUB Group, University of Washington, USA

Despite significant progress in the capabilities of mobile devices and applications, most apps remain oblivious to their users' abilities. To enable apps to respond to users' situated abilities, we created the Ability-Based Design Mobile Toolkit (ABD-MT). ABD-MT integrates with an app's user input and sensors to observe a user's touches, gestures, physical activities, and attention at runtime, to measure and model these abilities, and to adapt interfaces accordingly. Conceptually, ABD-MT enables developers to engage with a user's "ability profile," which is built up over time and inspectable through our API. As validation, we created example apps to demonstrate ABD-MT, enabling ability-aware functionality in 91.5% fewer lines of code compared to not using our toolkit. Further, in a study with 11 Android developers, we showed that ABD-MT is easy to learn and use, is welcomed for future use, and is applicable to a variety of end-user scenarios.

CCS Concepts: • **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; **User interface toolkits**.

Additional Key Words and Phrases: Ability-based design, accessibility, interface adaptation, mobile toolkit, developer tools.

ACM Reference Format:
Junhan Kong, Mingyuan Zhong, James Fogarty, and Jacob O. Wobbrock. 2024. The Ability-Based Design Mobile Toolkit (ABD-MT): Developer Support for Runtime Interface Adaptation Based on Users' Abilities. *Proc. ACM Hum.-Comput. Interact.* 8, MHCI, Article 277 (September 2024), 26 pages. https://doi.org/10.1145/3676524

1 Introduction

Mobile devices like smartphones and tablets have continued to rapidly develop, with significant improvements in capacitive touch screens and mobile sensors, enabling a wide range of capabilities and experiences. However, most mobile applications remain oblivious to the abilities of their users, displaying output and reacting to input in ways that generally treat every user the same [62]. For example, a user with fine motor challenges like tremor might have to adapt themselves to tap accurately on small targets, such as using a capacitive-tipped pointing stick. Despite device accessibility settings that allow customizations and preferences, research has found many

Authors' Contact Information: Junhan Kong, junhank@uw.edu, The Information School, DUB Group, University of Washington, Seattle, WA, USA; Mingyuan Zhong, myzhong@cs.washington.edu, Paul G. Allen School of Computer Science & Engineering, DUB Group, University of Washington, Seattle, WA, USA; James Fogarty, jfogarty@cs.washington.edu, Paul G. Allen School of Computer Science & Engineering, DUB Group, University of Washington, Seattle, WA, USA; Jacob O. Wobbrock, wobbrock@uw.edu, The Information School, DUB Group, University of Washington, Seattle, WA, USA.

ACM 2573-0142/2024/9-ART277
https://doi.org/10.1145/3676524

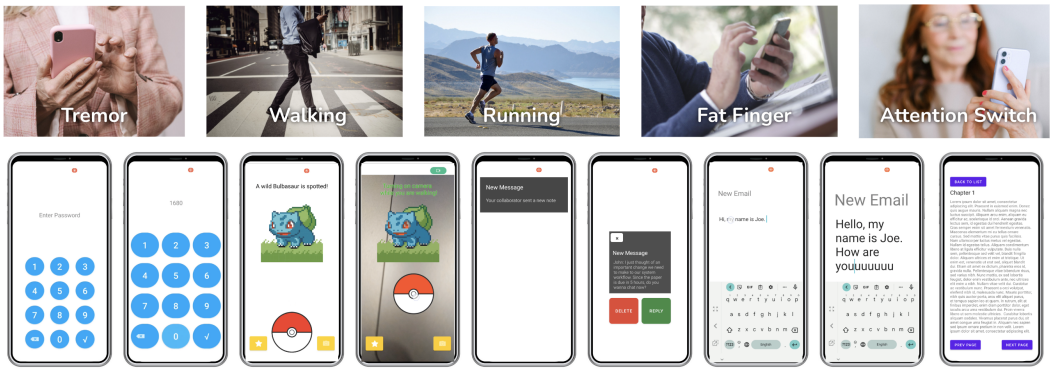

Fig. 1. Using the Ability-Based Design Mobile Toolkit (ABD-MT), a developer can make their application aware of and responsive to a user's abilities at runtime. Examples include UI widgets that respond to tremor, screen layouts that respond to walking and running, text editors that respond to fat fingers, and screen brightness that responds to attention switching.

users are unaware of these settings [11] or settings are not expressive or dynamic enough to accommodate their abilities-in-context [12, 61, 67]. Meanwhile, developers who wish to create accessible ability-aware mobile applications face significant challenges, as doing so requires extensive effort to process input and sensor data.

We therefore seek to provide developers of mobile applications a toolkit that supports making apps aware of and responsive to a user's situated abilities at runtime. To this end, we present the *Ability-Based Design Mobile Toolkit (ABD-MT)*. ABD-MT integrates with an app's input and sensor processing to enable developers to observe a user's behavior at runtime, model and reason about the user's abilities, and adapt interface widgets and layouts to better accommodate these abilities. In contrast to prior work that has measured and modeled abilities via tasks in an artificial test-bed to auto-generate interfaces which are then static [14–16], ABD-MT enables developers to monitor a user's abilities *at runtime* and to continuously adapt interfaces in response.

ABD-MT has three main modules: the *Observers*, the *Ability Modeler*, and the *UI Adapter*. Observers record input events and sensor data, currently for four types of abilities: touch, gesture, physical activity, and attention. Each Observer also provides ability information through calculation of human performance metrics. The Ability Modeler synthesizes captured interaction data and metrics from Observers to model a user's abilities, exporting methods for developers to inspect those abilities. For example, an app can query the Ability Modeler as to whether a user exhibits tremor, and whether this might be due to walking or running. Finally, the UI Adapter allows developers to manipulate UI widgets and layouts according to observed abilities and behaviors. Using ABD-MT's three modules, a developer can create an app that is aware of and responsive to a user's situated abilities at runtime. Conceptually, ABD-MT builds up a user's "ability profile" through observation of their behavior at runtime, then enables developers to engage with this profile to inform interface adaptations.

We evaluated ABD-MT through a combination of proof-of-concept applications, lines-of-code comparisons, and an empirical evaluation. We first built example applications to demonstrate the capabilities of the ABD-MT, how it supports a range of end-user scenarios, and how it does so with simplicity for developers. We then compare the amount of code required for each app with and without ABD-MT, and show a substantial reduction (91.5%) in lines of code. Additionally, we evaluate usage of ABD-MT through a study with 11 programmers who had Android development experience, further demonstrating that ABD-MT enables developers to create ability-aware apps

within a short amount of time, and that ABD-MT is easy to learn and use, is welcomed for future use, and is applicable to a variety of end-user scenarios.

The contributions of this work are: (1) an ability-based framework that observes user behavior, models user abilities, and enables adaptation of user interfaces, (2) an open-source implementation of that framework in a developer toolkit that can be used to build ability-based mobile apps, (3) an evaluation of the toolkit, including proof-of-concept example applications, lines-of-code comparisons, and an empirical evaluation with developers that demonstrates the usability and applicability of this toolkit.

2 Related Work

The core contribution of ABD-MT is a software framework that supports ability-based app development. ABD-MT is not an *automatic* user interface generator (e.g., SUPPLE [14]), but rather provides developers with easy ways to trigger interface adaptations *at runtime*, without need of test-bed tasks. In this section, we discuss the foundations of this work in Ability-Based Design [62, 63] and, more generally, in understanding user abilities. We also compare this work with context-aware computing [7, 8, 10] and automatic user interface generation [14–16].

2.1 Ability-Based Design

Our work builds upon Ability-Based Design (ABD) as first articulated by Wobbrock et al. [62, 63]. ABD is a design approach that focuses on users' abilities throughout the design process to create systems that "leverage the full range of human potential" [63]. An ability-based system takes upon itself the burden of accommodating a user's specific abilities, rather than a user having to adapt themselves to the system. ABD does not conceive of abilities only as properties of users in isolation, but as situated in a particular context, activity, or environment, as these things can significantly affect the exercisable abilities of the user. Therefore, ABD also considers "situational impairments" [53, 61], such as using a smartphone while walking [54] or in cold temperatures [39].

Example systems informed by ABD include WalkType [17] and ContextType [18], which adapt typing locations to detected walking and hand postures; SwitchBack [37], which aids reading resumption through gaze point detection; SUPPLE [14–16], which automatically generates interfaces optimized for reducing movement time and errors; work by Sarcar et al. [44–46] on optimizing interfaces for older adults; and recently, work by Alsaleem et al. [1] on using ABD to enhance outdoor play activities for kids.

To the best of our knowledge, our work is the first to provide a toolkit for developers to understand users' behaviors, infer their associated abilities, and adapt user interfaces accordingly, *all at runtime*. We position this work as yet another possibility for achieving the aims of ABD alongside automatic user interface generation, user-specific "design for one" approaches [20], end-user customization, and other forms of personalization.

2.2 Understanding User Abilities

There has been a great deal of work on understanding users' abilities. For example, MacKenzie et al. [36] devised pointing accuracy measures to characterize user pointing abilities and evaluate pointing devices such as mouses. Keates et al. [26] extended MacKenzie et al.'s measures to better understand pointing by people with limited fine motor function. Kong et al. [28, 29] proposed metrics to characterize touch abilities on touch screen devices. In this current work, the ABD-MT Touch and Gesture Observers utilize Kong et al.'s and MacKenzie et al.'s metrics when reporting touch and gesture behaviors.

Holz and Baudisch devised models [21, 22] to qualitatively characterize perceptual aspects of the touch process. Relatedly, Bi and Zhai [6] used a Bayesian criterion to model the probabilistic

distribution of touch selections, and Bi et al. [5] devised FFitts' law to model touch inputs on touch screens. These metrics further inform our Touch Observer module within ABD-MT.

Other work modeled user interactions with smartphones beyond on-screen touches and gestures. For example, a user's hand posture and pressure [18, 19], cognitive abilities [23] and psychomotor abilities [24, 25], walking activity [17], head movements [59], visual attention [31, 37, 42], and even blood alcohol level [38]. Mobile toolkits have also been created, for example, to extract time-of-day fluctuations of cognitive performance and detect patterns in users' alertness levels [9].

This prior research provides a rich set of techniques to sense and model users' situated abilities. Although not all techniques are currently used in this work, ABD-MT could incorporate such techniques into its framework through its modular architecture that combines observed behavior with ability modeling and UI adaptation.

2.3 Context-Aware Computing

There has been extensive discussion about a user's *context*, conceptualized as "the location, identity and state of people, groups and computational and physical objects" [8], and consisting of "the computing environment, the user environment, and the physical environment" [49]. There have been numerous context-aware systems that have been built. For example, Schmit et al. [51, 52] used mobile sensors to infer environmental parameters and device information. Barnard et al. [3] found that changes in motion, lighting, and task type affected user task performance. Dey et al. [8] created a toolkit for developing context-aware applications through separation of the "acquisition and representation" of context from the "delivery and reaction" to context. Schilit [50] proposed a mobile system that store dynamic environment information for multiple devices.

However, context-aware computing has not typically addressed users' abilities. This oversight has, in fact, given rise to accessibility issues [41, 65]. The ABD-MT work described herein focuses on the *abilities* of a user—therefore, the sensing and modeling of *activities* or *environments* do not attempt to explicitly model "context" in the way context-aware computing systems have done, but instead focuses on how context, such that it is sensed at all, ultimately affects a user's expressed behaviors at runtime.

2.4 Automatic User Interface Generation

With the goal of optimizing user interfaces to better suit users' abilities, systems have been created to generate the "optimal" user interface (UI) for a specific user's measured abilities. A seminal example is SUPPLE [13–16], which treats UI generation as an optimization problem, making UI selections to minimize pointing time which is parameterized by a user's performance in isolated test-bed tasks. Indeed, SUPPLE played a formative role in the conceptualization of ABD [63], through *design-time* UI generation. Relatedly, Macik et al. [35] proposed cross-platform automated UI generation through code inspection of native UI components. Peng et al. [43] recommended optimal accessibility settings by analyzing users' standard touchscreen gestures. Wu et al. [66] prototyped accessibility feature recommenders of UI parameters such as font sizes and side-button click-speed based on user input. Prior work has also explicitly applied ABD principles [62] to generate optimal UIs for different users and scenarios, such as supporting young adults with intellectual disabilities [4], developing play activities for children with special needs [55], creating adaptive outdoor experiences [1], and optimizing text entry for visual, motor, and cognitive disabilities [45].

In contrast these approaches, ABD-MT does not aspire to auto-generate user interfaces or provide *design-time* UI generation. Rather, it supports runtime UI adaptations – it provides developers a means to *encode* a set of rules to apply appropriate UI changes during *runtime*, in response to observed user abilities.

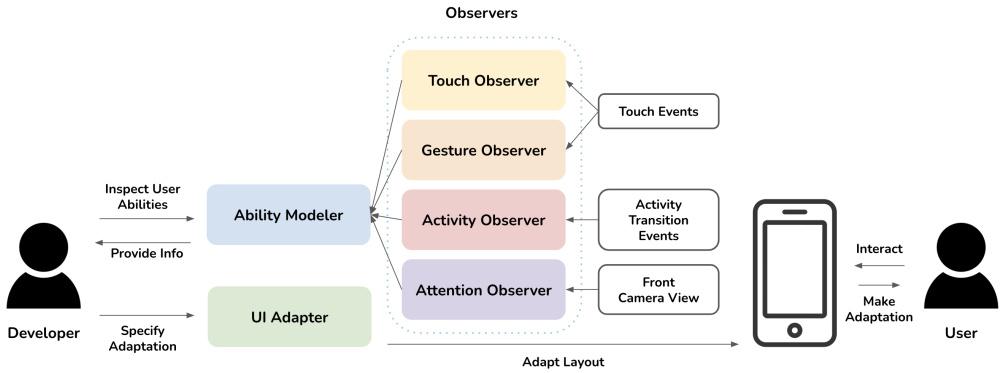

Fig. 2. ABD-MT architecture. (1) The *Observers* capture and record user touch, gesture, activity, and attention behaviors; (2) The *Ability Modeler* takes the observed user behavior, and it models and reasons about the user's abilities; (3) The *UI Adapter* allows developers to specify adaptations based on these abilities.

3 The Ability-Based Design Mobile Toolkit

ABD-MT combines three main concepts to collectively support efficient development of ability-aware applications: the *Observers*, the *Ability Modeler*, and the *UI Adapter* (Figure 2). We provide a table of all methods currently supported by our toolkit in Appendix B and additional documentation in our supplementary materials. ABD-MT was developed and runs on the Android platform. That said, the framework that ABD-MT establishes is not specific to Android. The three-part architecture of Observers, Ability Modeler, and UI Adapter is conceptually relevant to any mobile platform.

ABD-MT currently captures four dimensions of user abilities: touch, gesture, physical activity, and attention. Our choice of these four abilities aims at including a range of observable user behaviors, from low-level (e.g., on-screen touch and gestural traces) to high-level (e.g., situational information like the physical activity a user is engaged in), and at covering a variety of interaction modalities (e.g., touch-based interactions, bodily motion, eye gaze). Although we emphasize the potential of extending ABD-MT to account for additional abilities, this initial set was intended to demonstrate and explore ABD-MT's core capabilities.

3.1 Observers

As noted above, there are currently four *Observers* in ABD-MT: (1) the *Touch* Observer, which captures a user's on-screen touches like tapping; (2) the *Gesture* Observer, which captures a user's on-screen gestures like swiping and sliding; (3) the *Activity* Observer, which captures a user's physical activities including walking, running, cycling, driving, and staying still; and (4) the *Attention* Observer, which captures a user's attention including looking at and looking away from the screen.

3.1.1 Capturing User Interaction Data. The Observers each maintain a history of a user's touches, gestures, activities, and attention. Using Kong et al.'s [28, 29] modeling, each touch or gesture is represented as an array of `TimePoint` instances, consisting of (x, y) coordinates, contact area, major and minor axes, orientation, and timestamp of the oval (Figure 3). ABD-MT keeps track of the current unfinished touch or gesture path until it sees a finger-up event signaling the end of a trace, and then sends the complete path to the Android gesture classifier to determine if it should be recorded by the Touch or Gesture Observer.

Physical activity and attention are represented using the `Activity` and `Attention` instances, each containing a name, starting time, and ending time (Figure 3). To record a user's physical activities and attention, ABD-MT wraps Android's Activity Recognition Transition API [32] and

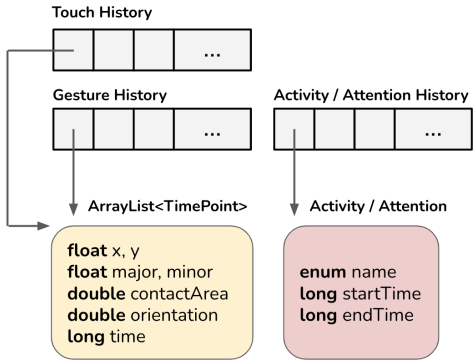

Fig. 3. *Observer* data types and data structures. ABD-MT currently implements Observers for touch, gesture, physical activity, attention, but in principle could be extended to any number of user behaviors or situations.

Google's Vision API [33], respectively. When an *entering* activity transition event occurs, the ABD-MT records its `startTime`, and when an *exiting* activity transition event occurs, it records the `endTime` and passes the completed activity to the Activity Observer. When a user's eyes appear or disappear from the front camera view, indicating that the user starts looking at or looking away from the screen, ABD-MT records the `endTime` of the previous attention status, passes it to the Attention Observer, and records the `startTime` for the current attention span. Although these default APIs already provide *some* information regarding a user's physical activities and attention, our Observers further allow the Ability Modeler to combine input from multiple Observers to reason about higher-level abilities and across different aspects of a user's abilities (Section 3.2).

3.1.2 Exposing Human Performance Metrics. In addition to capturing user behaviors, the Observers allow developers to access a rich set of low-level human performance metrics characterizing user behaviors. The touch metrics exposed include touch direction, variability, drift, duration, extent, area change, area variability, area deviation, angle change, angle variability, and angle deviation [28, 29]. The gesture metrics include movement variability, movement error, and movement offset [36]. The activity and attention metrics include the total duration of each activity or attention and the uptime of the current activity or attention. These metrics can be directly used by developers to inform user-specific adaptations, but are also utilized by the Ability Modeler as it forms a holistic understanding of a user's abilities.

Each Observer API method provides different overloads for querying the metrics with different conditions. For example, the touch drift metric can be calculated from all touches as `getTouchDrift()`, from the last N touches as `getTouchDrift(lastN)`, from touches within a time range as `getTouchDriftMs(startTime, endTime)`, and from touches since a starting time as `getTouchDriftMs(startTime)`. Similar methods are offered by the other Observers.

3.2 Ability Modeler

ABD-MT's *Ability Modeler* takes the observed user behavior as well as the metrics calculated in the Observers and synthesizes these inputs to model user abilities. The Ability Modeler not only provides user ability information for each *Ability Group*—fine motor abilities, physical activities, and attention, but it also provides insight into how abilities are affected by situational factors (*e.g.*, "situational impairments" [53, 61]) and the interaction of different *Ability Groups*—for example, whether observed tremor might be due to walking, or whether a user is looking at their phone while they are doing some physical activity.

3.2.1 Fine Motor Ability Group. The Ability Modeler provides information regarding a user's fine motor abilities through methods like `hasTremor()` and `hasStiffness()`. Based on prior work showing that touch variability is significantly related to specific fine motor challenges including tremor [28, 29, 34], `hasTremor()` checks whether the touch variability (Section 3.1.2) is above a threshold. Similarly, `hasStiffness()` checks whether the touch drift (Section 3.1.2) is above a threshold without having a touch variability that is too large.

3.2.2 Physical Activity Ability Group. The Ability Modeler offers methods to query a user's physical activities, including `isWalking()`, `isRunning()`, `isCycling()`, `isDriving()`, and `isStill()`. It returns whether the user is currently engaged in the activity; the ABD-MT also provides a set of listeners that can enable an app to respond immediately when an activity is detected, for example, using `onWalk()`.

3.2.3 Attention Ability Group. The Ability Modeler offers methods to query a user's attention through `isLooking()`, which queries the Attention Observer and returns whether the user is currently looking at the screen. Similar to activities, the Ability Modeler provides listeners enabling an app to respond immediately to an attention switch, using `onLook()` and `onLookAway()`.

3.2.4 "Situational" Fine Motor Abilities. The Ability Modeler also provides information about a user's "situational" fine motor abilities, by inspecting the interaction between the *Fine Motor Ability Group* and the *Physical Activity Ability Group*. The `hasWalkingTremor()`, `hasRunningTremor()`, and other similar methods query average touch metrics during the time intervals of the specified activity (Section 3.1.1), determining tremor in a similar fashion as `hasTremor()` (Section 3.2.1). Similar to querying the metrics from the Observers, the Ability Modeler also provides different overloads for querying overall and situational fine motor abilities within a certain time range or subset of all recorded interactions (Section 3.1.2).

3.2.5 Additional Interaction Between Ability Groups. The Ability Modeler offers methods to query interactions between physical activities and touch and gesture behaviors, through `isWalkingWhileTouching()`, `isWalkingWhileUsingGestures()`, and similar methods for running, cycling, and driving. These methods query the Activity Observer for the current activity and its starting time (Section 3.1.1), and then the Touch or Gesture Observer to check if any touch or gesture has been performed during that time. The Ability Modeler also offers methods to query interactions between attention and touch, gesture, and physical activities, through `isLookingWhileWalking()` and similar methods for running, cycling, driving. These methods query the Attention Observer for the current attention status and its starting time (Section 3.1.1), and then the Touch, Gesture, and Activity Observers to check if any touch, gesture or activity is performed during that time.

3.3 UI Adapter

The *UI Adapter* keeps a record of all user interface (UI) widgets and exposes methods for a developer to manipulate multiple UI widgets (`View` instances) to change the look of their applications.

3.3.1 Registering UI Widgets. The UI Adapter maintains a hashset of all widgets. The UI Adapter iterates through all children of the root widget (`ViewGroup`) to register them upon initialization of each application screen, and when any widget programmatically created during runtime, for example, contents of a `RecyclerView`. In this way, the UI Adapter stores references to all of an app's widgets to enable adaptations.

3.3.2 Changing Widget Sizes. An important ability-based UI adaptation for mobile applications is changing widget sizes. The UI Adapter provides the `resizeWidgets(factor)` and the

`enforceMinSize(bounds)` methods, and similar methods for font sizes. To achieve this, the UI Adapter iterates through all widgets that have been registered with the toolkit, applying the scaling factor to each widget.

3.3.3 Showing, Hiding, and Activating Widgets. In addition to changing UI widgets, the UI Adapter also enables manipulation of a subset of widgets, organized by their `tags`. The `tag` of each widget can be specified through the `setTag(view, tag)` method.[1] Based on this tagging capability, the UI Adapter can show or hide multiple related widgets based on their tags through methods like `showWidgetsWithTag(tag)` and `hideWidgetsWithTag(tag)`; for example, a developer can tag certain widgets as "important" and show only those when a simpler UI is desired. The UI Adapter also exposes a `sendClick(view)` method to programmatically perform a click (tap) on a widget, and a `showAlert(message)` method to create and display an alert message on the screen.

3.3.4 Changing Screen Brightness. The `changeBrightness(activity, factor)` method changes overall brightness of the screen.[2] This method applies a scaling factor to the current screen brightness.

3.3.5 Additional Variations of UI Adaptations. For a number of adaptations above, more variations are available in the UI Adaptor to provide better flexibility for developers. A developer can specify the `tag` of widgets to be manipulated (see Section 3.3.3). A developer can also specify the parent widget and apply the changes only to its children, for example, `resizeWidgets(factor, parent)`. In order to prevent frequent and undesired changes, the UI Adapter provides an additional option to apply adaptations based on the initial configurations of widgets through methods like `changeBrightness(factor, basedOnInitialValue)`. For example, a developer can change the screen brightness to 1.5 times the original, but not beyond.

3.3.6 Reverting UI Adaptations. The UI Adapter also keeps a history of adaptations made, including the type of adaptation and a mapping of each widget to their UI parameters. With this record, the UI Adapter exposes a `revertUI()` method to revert the user interface to prior configurations.

```
"transformations": [{
    "type": "Button",
    "transformation": [{
        "transformationAttribute": "Width",
        "transformationFactor": 1.5
    },{ "transformationAttribute": "Height",
        "transformationFactor": 1.5
    }]}, {
    "type": "TextView",
    "transformation": [{
        "transformationAttribute": "TextSize",
        "transformationFactor": 1.2
    }]}]
```

Fig. 4. Example JSON code that updates all buttons to have 1.5 times their original size and all text strings to use 1.2 times their original font size.

[1]Note that the `setTag()` method in the UI Adapter is different from the existing Android `setTag()` method in that the former is not designed to store extra pieces of information for each view, but instead provides a group identifier for showing or hiding multiple widgets at once.

[2]Note that the `activity` parameter here is the current Android activity (i.e., the running application), and not related to the Activity Observer.

3.3.7 Writing UI Adaptations Using JSON. When developers wish to apply multiple UI changes to a set of widgets of the same type, they can specify these changes in a JSON file similar to style sheets on web pages (Figure 4) using the `loadUITransformation(activity, resourceId)` method by specifying the `resourceId` of this JSON file.

3.3.8 Switching UI Layouts. In some cases, developers might not want to adapt individual UI widgets, but rather switch to new UI layouts altogether. In such cases, the `loadUISource(page, layoutId)` method provides a means to load and display alternative UI layouts at runtime. Such files are already familiar components of so-called "responsive apps," which utilize different layouts for different device screen sizes.

3.4 Client Interface for Using ABD-MT

For a mobile app developer to use ABD-MT, they first create an Android project in Android Studio, importing the ABD-MT as a module and adding required lines in configuration files.[3] In each activity's source code file (e.g., `MainActivity.java`), the developer replaces the default `AppCompatActivity` with `ABDMTActivity` from which the main activity inherits and initializes the toolkit by obtaining necessary toolkit instances, activating the Observers, and registering UI widgets (Figure 5, left). After these initializations, developers can use the toolkit's API calls to add ability-based functionality to their apps (Figure 5, right).

Initialization

```
class MainActivity extends ABDMTActivity {
 ... // Declare singleton variables
 @Override
 void onCreate(Bundle savedInstanceState) {
  ... // App initialization code
  setContext(this);
  setRoot(findViewById(R.id.root));
  _abdmt = ABDMT.getInstance();
  _abdmt.startObserver(ObserverFlags.TOUCH);
  _touchObserver = _abdmt.getTouchObserver();
  ... // Same for gesture/activity/attention
  _abilityModeler = _abdmt.getAbilityModeler();
  _uiAdapter = _abdmt.getUiAdapter();
  _uiAdapter.registerWidgets(getRoot());
}}
```

Example Runtime Uses

```
if (_abilityModeler.hasTremor()) {
 double b = _touchObserver.getTouchExtent();
 _uiAdapter.enforceMinSize(b);
}
...
if (_abilityModeler.isRunning()) {
 _uiAdapter.showOnlyWidgetsWithTag("important");
}
...
if (_abilityModeler.hasWalkingTremor()) {
 _uiAdapter.resizeFonts(1.5, true);
} else {
 _uiAdapter.resizeFonts(1, true);
 _uiAdapter.revertUI(); // alternatively
}
```

Fig. 5. A developer's initialization (left) and example uses (right) of ABD-MT. The left code block initializes ABD-MT inside each Android activity, replacing "AppCompatActivity" with "ABDMTActivity" as the base class the main activity inherits. Nothing more is required. The right code block includes example uses of the toolkit's API for runtime adaptation. For instance, the third example implements a UI that enlarges fonts by 50% when the user is walking *and* exhibits signs of tremor, and reverts fonts to their original size otherwise.

4 Evaluation

4.1 Goals of Evaluation

To evaluate ABD-MT, we first built five example applications and compared their lines of code with and without the toolkit, then conducted a study with 11 Android developers. We structure our evaluation following common toolkit evaluation methods [30], emphasizing *demonstration* through our example applications and *usage* through our study. Specifically, we first *demonstrate* usage scenarios of the toolkit through five example application case studies (Section 4.2, "case studies" [30]).

[3]We make the ABD-MT source code and example app implementations available at the following GitHub repo: https://github.com/abdmt-toolkit/abdmt. The full documentation is also available at https://github.com/abdmt-toolkit/abdmt/wiki.

We then further evaluate the toolkit *usage* by combining four common methods of toolkit usage evaluation suggested in prior work [30]: (1) A lines-of-code comparison of the five example apps we built in Section 4.3 ("A/B comparisons" [30]); (2) Task completion times and subjective ratings during a study with 11 Android developers in Section 4.4 ("usability study" [30]); (3) Apps *designed and implemented* by participants after the lab study in Section 4.4.4 ("take-home studies" [30]); (4) Additional apps *ideated* by participants during the study in Section 4.4.4 ("observation" [30]).

Section 5.4 also discusses how our toolkit addresses the seven principles of Ability-Based Design and the toolkit's architectural contributions. Note that we did not conduct a technical evaluation of accuracy in ABD-MT's current ability detection, as they are either already evaluated in prior work (touch [29] and gesture [36]) or are state-of-the art detection techniques (activity [32] and attention [33]).

4.2 Example Applications

To demonstrate the capabilities of ABD-MT, we first built five example apps using the toolkit.

4.2.1 Example 1: Enlarging Keyboard Due to Tremor. For some users or in some situations, the keys of a virtual keyboard can be too small to tap accurately [64]. Using ABD-MT, we built a password-entry keyboard that automatically enlarges according to the presence and amount of a user's tremor. The keyboard app first calls the *Ability Modeler*'s `hasTremor()` method upon any touch input. If the user seems to exhibit tremor, the app calls the Touch Observer's `getTouchExtent()` method to ask for the average touch extent, or "2-D touch footprint," of the user [29]. The keyboard keys are then enlarged to be four times the user's average touch extent,

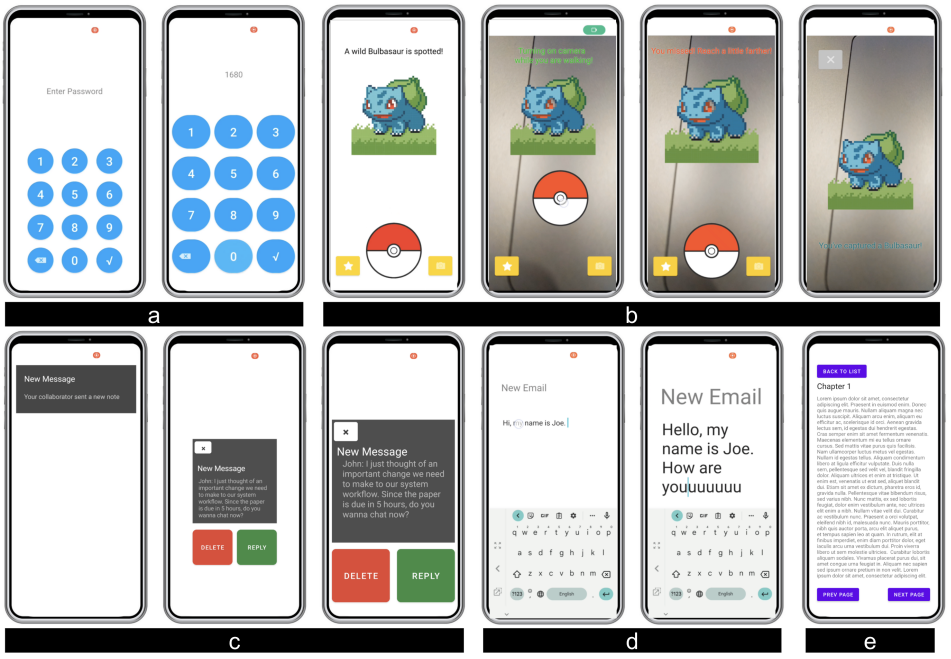

Fig. 6. Example apps created using ABD-MT: (a) a password-entry keyboard that enlarges proportionally to the amount of tremor detected; (b) an adaptive Pokémon Go game that brings targets closer and turns on camera for situational awareness during walking; (c) running-aware text notifications that responds to walking through added buttons and enlarged widgets and fonts; (d) a zooming text editor that makes it easier for users to finely position a text cursor between two characters; (e) a reading app that changes screen brightness based on user attention.

utilizing the `enforceMinSize()` method from the *UI Adapter*. In contrast to accessibility guidelines emphasizing a single minimum target size, this example illustrates how the ABD-MT can support adaptation to individual motor ability.

4.2.2 Example 2: Adaptive Pokémon Go for Walking. Walking can create situational impairments that affect how proficiently a person can operate a mobile user interface [61]. We built an adaptive version of Pokémon Go that responds to a user's gestures while they are walking (Figure 6b). The app is similar to the traditional *Pokémon Go* game, in which a user uses a flick gesture to throw a Pokémon ball towards a Pokémon creature. In our adaptive version, the Pokémon creature is brought closer to the Pokémon ball when the user's gesture is repeatedly insufficient to reach the creature, which occurs often during one-handed use [19]. In addition, a Pokémon Go user can turn on the phone camera for an augmented reality (AR) experience that shows the creature overlaid on the real world. Our adaptive version automatically detects when the user is walking while trying to perform on-screen gestures and triggers the camera so that the user can see their surrounding environment. Whenever a user swipes anywhere on the screen, the app checks with the *Ability Modeler* using the `isWalkingWhileUsingGestures()` method to know whether this user is walking while trying to perform a gesture to catch the Pokémon creature. If so, the app programmatically clicks on the camera button through the *UI Adapter*'s `sendClick()` method and brings the Pokémon creature closer to the Pokémon ball to let the user more easily reach it. This app illustrates how ABD-MT allows apps to accommodate affected abilities of mobile interactions in the presence of "situational impairments."

4.2.3 Example 3: Running-Aware Text Notifications. Even more than walking, running introduces the possibility of situational impairments [61]. We built an app that shows different UI layouts and widget sizes to display notifications in response to a user's running activity (Figure 6c). When a text message arrives, the app displays a typical small notification atop the screen. If the user is running, the app displays a larger notification window with more detailed content. The app keeps enlarging the message and buttons if the user fails to accurately dismiss the message. The widgets that make up the minimized notification (Figure 6c, left) and the widgets that make up the larger notification window (Figure 6c, middle) are tagged as "small" and "large", respectively, using the UI Adapter's `setTag()` method. When a user is detected as running, the `onRun()` listener is triggered, and the client app calls `showOnlyWidgetsWithTag("large")` to display the large notification window (Figure 6c, middle). If the user's subsequent touches fail to handle the notification, the window continues to enlarge using the `resizeWidgets()` and `resizeFonts()` methods in the *UI Adapter* to make the buttons easier to tap while running (Figure 6c, right). This app not only demonstrates how ABD-MT recognizes a user's situation, but further how it supports adaption of multiple interface widgets (using "tags"), instead of needing to adapt each individual element.

4.2.4 Example 4: Zooming Editor. We built a text editor that accommodates the "fat finger" problem [21, 57, 60], specifically when users have trouble placing their text cursor precisely between characters. In our example app (Figure 6d), a user can hold their finger on the screen for a brief duration, causing the text editor to show progressively larger fonts, thereby enabling the user to easily place their text cursor between desired characters. After the user places the text cursor, the fonts automatically shrink back to their original sizes. Our zooming text editor uses the *Touch Observer*'s `getTouchDuration()` method to check whether the user touches for longer than 500 ms; if so, the *UI Adapter*'s `resizeFonts()` method is used to set the font size to be 1.5 times the original font size. After the text cursor has been placed and the finger lifts, the app calls the *UI Adapter*'s `revertUI()` method to restore the original font size. This app illustrates how ABD-MT

	Enlarging Keyboard Due to Tremor		Adaptive Pokémon Go for Walking		Running-Aware Text Notifications		Zooming Editor for Fat Fingers		Adaptive Reading Brightness	
	With Toolkit	Without Toolkit	**With Toolkit**	Without Toolkit	**With Toolkit**	Without Toolkit	**With Toolkit**	Without Toolkit	**With Toolkit**	Without Toolkit
Ability-Based Functionality	**3**	162	**8**	139	**24**	174	**10**	88	**16**	162
Total	**121**	267	**214**	327	**60**	192	**38**	103	**36**	174

Table 1. Comparison of the number of lines of code for each example app with and without our toolkit.

maintains a history of adaptations and enables a user to revert adaptations that are unwanted or no longer needed.

4.2.5 Example 5: Adaptive Reading Brightness. When reading on mobile devices, a user needs enough screen brightness to see the text. However, they might not want to always keep the screen bright. We built a reading app (Figure 6e) that increases the screen brightness when a user looks at the screen, but dims the screen when the user looks away, similar to Apple's Attention Aware feature [2]. When a user is detected to start looking at the screen by the *Attention Observer*, the `onLook()` listener is triggered, and the client app calls the `changeBrightness()` method from the *UI Adapter* to increase the brightness. When the user looks away, the `onLookAway()` listener is triggered, reverting the screen to its original brightness using `revertUI()`. This app illustrates additional capabilities of ABD-MT, and additional modalities it supports.

4.3 Lines of Code Reduction

We built each example application with and without our toolkit, then counted (1) the lines of code required to achieve the ability-aware functionality, and (2) the total lines of developer code required for the entire application.[4] We present the results in Table 1 and Figure 8a. The ability-based capabilities of these examples were all achieved using only a few lines of app-specific code, plus typical initialization code ranging from 8-18 lines. By comparison, achieving the ability-based functionality without ABD-MT required 162, 139, 174, 88, and 162 lines of code for each example. On average, ABD-MT provided a 91.5% (*SD*=4.7) savings in lines of code dedicated to ability sensing and UI adaptation.

4.4 Study with Android Developers

To further evaluate ABD-MT, we conducted a study with 11 Android developers. We further demonstrate the usability[5] of the toolkit through **task completion time** of implementing four ability-based features and through **subjective ratings**, and the expressiveness of the toolkit through **apps that participants *designed and implemented*** plus **additional designs that participants *ideated***.

4.4.1 Participants. We recruited 11 participants from the University of Washington who have previously developed Android apps using Android Studio through website postings and email

[4]A complete application can of course require thousands of lines of code and numerous additional libraries, themselves each containing thousands of lines of code. By "developer code," we mean only the code that an app developer writes to make each of our example apps functional in the ways we describe. A complete application can also have multiple possible implementations. The lines of code reported in our evaluation are counted according to our understanding of the most concise and simple implementation. The lines of code numbers were counted with *cloc* (https://github.com/AlDanial/cloc), excluding package names, imports and blank lines.

[5]Here, the term *usability* specifically refers to the usability of the toolkit in providing development support for ability-aware apps. The end-user usability of apps created by developers is not evaluated in this study.

Participant Id	Gender	Age	Time Spent (Hours)	Apps Developed
P1	Man	21	0-10	1
P2	Man	22	>80	8
P3	Man	24	40-80	2
P4	Woman	22	10-40	2
P5	Woman	24	10-40	2
P6	Woman	25	>80	6
P7	Man	26	10-40	1
P8	Man	25	40-80	3
P9	Man	19	40-80	6
P10	Non-Binary	21	40-80	6
P11	Man	20	>80	15

Table 2. Demographics of the 11 participants who had Android development experience. "Time Spent" includes the number of hours each participant had spent in total on Android development; "Apps Developed" indicates the number of apps each participant had previously developed.

solicitations. Table 2 shows participant demographics. On average, participants were 22.6 years old (SD=2.3), self-reported identifying as 3 women, 7 men, and 1 non-binary person. Participants had varying levels of Android development experience, with 1 participant having 0-10 hours of total experience, 3 having 10-40 hours of total experience, 4 having 40-80 hours of total experience, and 3 having >80 hours of total experience. On average, participants had developed 4.7 (SD=4.2) Android apps. We discuss our participant choice, its implications and limitations in Section 6.

4.4.2 Apparatus. The ABD-MT toolkit library and documentation were distributed to participants through a GitHub repository prior to the study. We also included scaffold code of (1) a tutorial app with code comments explaining the steps (Figure 7c), and (2) a todo-list app,[6] which is a common starter application in Android development courses (Figure 7b). Participants had the option to use their own laptop with Android Studio (Electric Eel) installed before the study session, or to use a laptop provided by the research team. Participants also had the option to use their own test Android phone, or to use a provided Google Pixel 4a test phone.

4.4.3 Procedure. Study sessions were conducted in-person with approval from our university's Institutional Review Board. After a brief introduction of the study and the toolkit, participants first went through a 20-minute tutorial session to complete a simple tutorial app (Figure 7a) to learn our toolkit. Participants followed the documentation to add the ABD-MT library to the tutorial app, and then wrote the click handlers for the buttons shown in Figure 7a to each display the average duration and variability of all touches captured and whether the user has tremor, and enlarge all the button sizes by twice.

Participants were then asked to make an existing app (the todo-list app) ability-aware, first by adding the ABD-MT library, then by writing four experimenter-specified ability-aware features: (1) to enlarge all widgets by 50% if a user has tremor, (2) to enlarge all fonts by 20% when a user is looking at the screen, and to revert back when the user looks away, (3) to make all widgets have minimum width and height of 10 times the average touch extent, and (4) to display only the existing

[6]The todo-list app was not created by the research team, but rather used an example implementation available online (https://itsourcecode.com/android-projects/best-to-do-list-app-on-android-with-source-code/) to reflect real-world usage of applying the toolkit to existing applications.

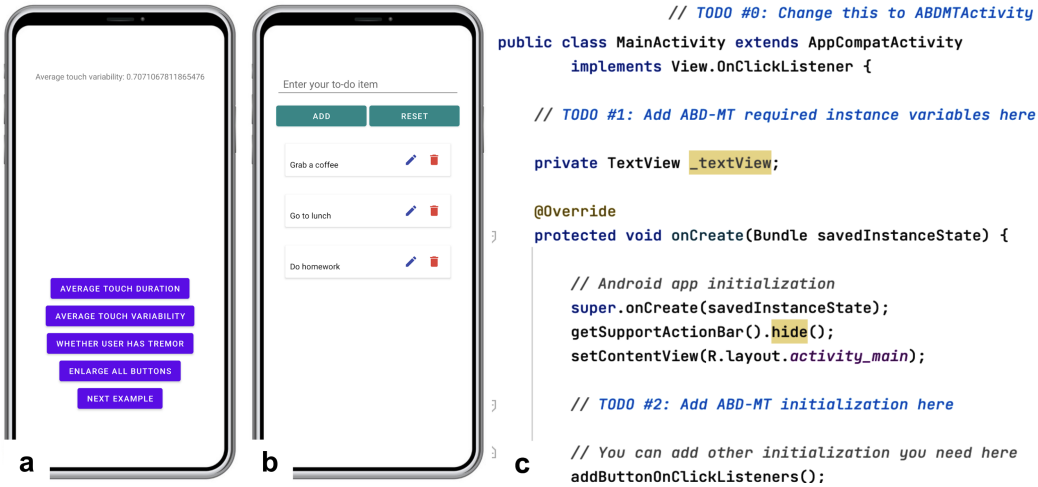

Fig. 7. Screenshots and code snippet of scaffold code provided for participants during the study: (a) a screenshot of the tutorial app, containing a text view and buttons specifying varying ability-based functionality, (b) a screenshot of the todo-list app with "add" and "reset" buttons and a text editor, as well as three todo-list items, and (c) code snippet of the tutorial app scaffold code, containing comments specifying locations to add tutorial code.

todo's when a user is walking. Participants could optionally spend up to 8 hours in the week after the session to build an ability-aware app of their own choice and design. After they were done with their development, participants were asked about the experience in a 30-minute semi-structured interview to rate the experience, to provide open-ended feedback, and to brainstorm other ability-aware apps that could potentially be built using the toolkit. Participants could optionally submit up to two other usage scenarios for the toolkit after the interview. Each study session took 2 hours in total, and participants were compensated at the rate of $20 per hour. Participants were compensated $10 for each additional use case they submitted.

4.4.4 Results. We present our study results in four aspects as discussed in our goals of evaluation (Section 4.1): (1) task completion time, and (2) subjective usability ratings and feedback, (3) example apps that participants designed and implemented using the toolkit after the study session, and (4) additional example apps that participants ideated as appropriate based on their experience with ability-based design in ABD-MT.

Task Completion Time. All 11 participants successfully completed the tutorial and developed the four experimenter-specified adaptations. During the tutorial sessions, participants spent an average of 7.6 minutes (SD=2.0) completing the toolkit set up and initialization, and 9.9 minutes (SD=2.7) writing the button click handlers, with an average total time of 17.5 minutes (SD=4.0) spent in the tutorial session. During the development session, participants spent an average of 5.2 minutes (SD=1.4) setting up and initializing the toolkit, 4.3 minutes (SD=2.0) completing the first experimenter-specified adaptation, 3.5 minutes (SD=1.1) completing the second adaptation, 3.1 minutes (SD=1.1) completing the third adaptation, and 7.8 minutes (SD=2.5) completing the fourth adaptation, with an average total time of 23.8 minutes (SD=4.7) spent in the development session.

Subjective Ratings. Overall, participants provided very positive feedback on the usability of the toolkit in enabling easy development of ability-aware apps and providing good coverage of needed functionality. Participants found the toolkit easy to learn (M=2.1, SD=0.7) and easy to use (M=2.0,

SD=0.9). Specifically, participants described the toolkit as *"very straightforward," "very intuitive," "easy to understand,"* and *"easy to use."* Participants also found the toolkit quite *"modularized"* (P1, P7) and they could *"easily incorporate it into the development process"* (P3, P7, P10):

> *"It's incredibly easy to use."* –P10
>
> *"It was pretty simple and straightforward."* –P9
>
> *"Before I saw the methods, I was like 'that sounds very hard'...it was surprisingly straightforward. There's very little code needed."* –P6

Participants said they enjoyed working with the different modules of the toolkit. For example, P5 *"like[d] the whole ability modeler"* and found it *"really interesting,"* and P2 *"like[d] the idea of using the UI adapter to change the display."* Participants also said it covered the most important aspects of abilities they needed:

> *"It's very helpful. It has very cool features."* –P4
>
> *"There are definitely a lot more modalities to imagine and things that can be done, eventually, but I think it [the toolkit] covers the most important ones."* –P5

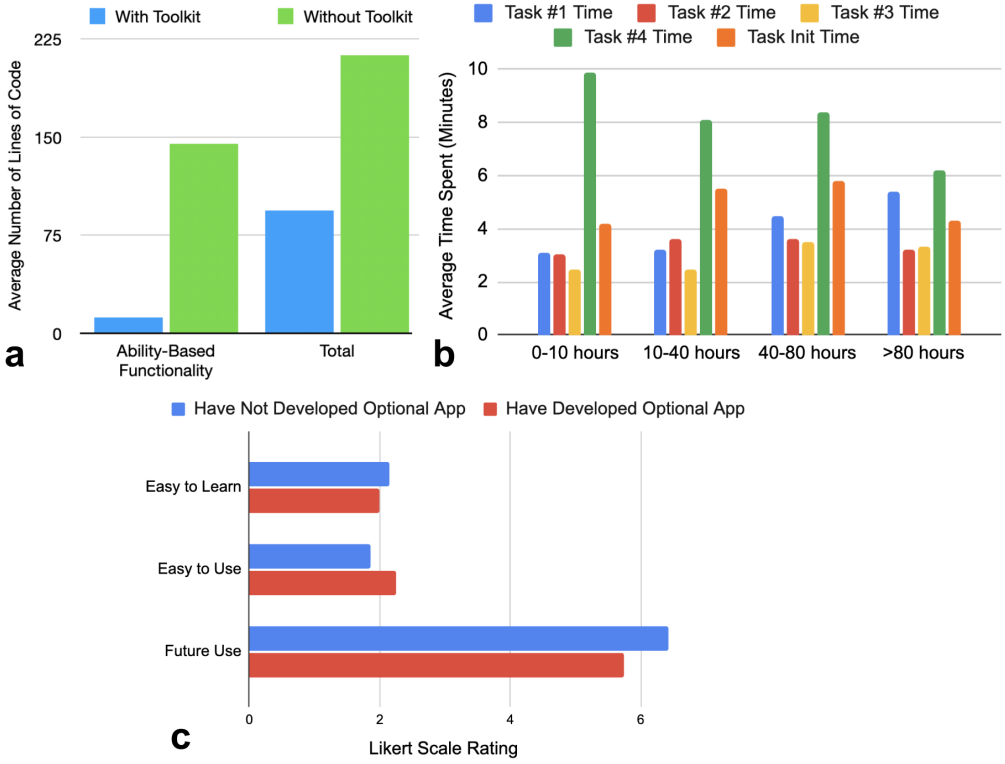

Fig. 8. (a) Average number of lines of code required for the five example apps with and without the toolkit, required for the ability-based functionality and for the entire app; (b) average time taken (minutes) to write up each adaptation, for developers with 0-10, 10-40, 40-80, and >80 hours of Android development experience—participants took a short amount of time to complete each task, regardless of prior Android experience; (c) Likert (1-7) ratings of the toolkit—overall, whether or not participants developed the optional app, they found the toolkit easy to learn and use (lower is better), and would consider using it in the future (higher is better).

> *"There are a lot of features, basically what you want is all there; I can't think on top of my head anything that is missing."* –P8

Participants expressed interest in ABD-MT and said they would consider using the toolkit to build ability-aware apps in the future (M=6.2, SD=1.3):

> *"I'll for sure consider it! It's great."* –P3

> *"The library offers enough functionality; I would be excited to use it, even thinking about universal audiences, regardless of what app I'm building and even it doesn't have accessibility focuses...I see a lot of potential in it."* –P10

Apps Designed and Implemented by Participants. Of the 11 participants, 4 chose to spend extra time after the lab study to design and implement an ability-aware app of their own choice. The resulting apps included:

- P2: A music player that enlarged its album cover images when a user started walking, and reverted them to their original size when the user stopped.
- P3: A messaging app that hid older messages and made the text message editor and "send" button full-screen when a user was walking, and restored all older messages when the user stopped.
- P4: A screen-time monitoring app that dimmed the screen when a user looked at the display for an extended period of time.
- P9: A todo-list app that hid all the content when a user was driving, and brought back all the widgets when the user stopped.

Participants self-reported spending an average of 2 hours and 50 minutes (SD=1 hour and 47 minutes) developing their apps. Although we were not able to accurately track the time each participant spent on different parts of their development (e.g., ideation, Android programming and debugging, searching toolkit documentation), participants self-reported spending the majority of time on "basic Android stuff" (P3, P9), and overall found the toolkit functionality easy to incorporate.

Additional Apps Ideated by Participants. Aside from the apps participants designed and implemented, participants also came up with a total of 69 additional ability-aware apps that they proposed can be built with the toolkit (62 after excluding duplicates). The proposed apps span a variety of categories, including music and video players (10), maps (8), reading (5), social media (5), messaging (4), games (3), calendar (2), creativity tools (2), health tracking (2), safety monitoring (2), productivity monitoring (2), banking (1), browsers (1), development tools (1), emails (1), farm management (1), learning (1), notebooks (1), phone calls (1), spreadsheets (1), video conferencing (1), and general adaptations without a specific application context (8).

Among the 62 proposed apps, 85.5% addressed the types of abilities captured by our Observers (25.8% addressing touch and/or gesture abilities, 50.0% addressing physical activities, and 21.0% addressing attention), and 16.1% required additional abilities to be observed. These additional Observers would need to detect low vision, posture and grip, gaze location, face proximity to the screen, whether a user has frequent misses, whether a user wears glasses, lighting conditions, noise levels, and things about the location and environment.

On the type of adaptations the proposed apps make, 83.9% utilized our UI Adapter methods to make interface adaptations, and 24.2% required additional adaptations beyond the current toolkit capabilities. These adaptations include smoothing a wiggly line drawn, picking up a call or sending a message to a doctor or emergency contact, changing volume or sound duration, locking the screen, enabling audio input and output, and applying customized gestures for app control.

On different types of impairments addressed by proposed apps, participants considered users with various forms of disabilities including people with Parkinson's disease, people with ADHD, older adults, blind or low-vision (BLV) users, as well as "situational" abilities affected by movement or distraction. A full list of the ideas participants brainstormed is included in Appendix C.

5 Discussion

We discuss the implications of ABD-MT, and in particular focus on how it simplifies the development of runtime-responsive ability-based apps, the variety of applicable user scenarios, and how it enables real-time interface adaptations. We also discuss its extensibility and highlight its toolkit contributions.

5.1 Simplifying Development of Ability-Aware Mobile Applications

ABD-MT significantly simplifies the development of runtime-responsive ability-based apps—ability-based functionality was added to each app with only a few lines of code, requiring only 3-24 lines of app-specific code, compared to 88-174 lines without the toolkit (a 91.5% average savings). Study participants took an average of 3.1 to 7.8 minutes to implement each ability-aware feature. This small amount of time required for development was consistent across the participants that were more or less experienced in Android development. In addition to the reduction in lines of code and the small amount of time required for development, subjective ratings also show that ABD-MT is easy to learn (M=2.1) and use (M=2.0), and of interest for future use (M=6.2).

5.2 Addressing a Variety of User Scenarios

In addition to simplifying ability-based app development, ABD-MT can sense and expose a range of situated abilities and facilitate corresponding interface adaptations. Our five example applications addressed abilities including tremor, walking, running, fat fingers, attention switching, and combinations of abilities. In addition, developers described the toolkit as covering the majority of "important" abilities and adaptations they needed, and it was already capable of supporting the four apps participants designed and implemented plus a large number of the additional ability-based apps they ideated (i.e., addressing 85.5% of ideated apps' ability sensing and 83.9% of ideated apps' adaptations).

ABD-MT addresses traditional accessibility use cases (e.g., tremor) and impairing situations (e.g., from physical activity). In its current accessibility use cases, ABD-MT focuses on motor impairments, specifically fine motor challenges which can be characterized by touch metrics obtained through user input behavior [28, 29]. Furthermore, the apps ideated by our participants suggest ABD-MT can potentially be extended to address a greater range of disabilities (e.g., low vision, ADHD). On the other hand, ABD-MT also considers abilities as "situated" under circumstances such as physical activities, with the potential of expanding to additional situational factors (e.g., noise, lighting, location, and temperature). Although we separately discuss motor impairments and situational impairments [61], they can together inform a holistic understanding of a user's need for runtime adaptation. To address these issues, there is a large space of potential apps that can be created with our toolkit, and participants said that potentially *"any app can benefit from such adaptability"* (P4, P11).

Together with the previous section, we demonstrate that our toolkit is both (1) *usable*, in that it is easy to use and welcomed by developers, significantly reduced the required lines of code, and only required a short amount of time to learn and use; and (2) *expressive*, in that it supports a variety of user scenarios and that developers could already implement various different ability-aware features and apps using it.

	How ABD-MT Addresses Each Principle of Ability-Based Design
Ability	The toolkit observes user behavior and provides a holistic understanding of user abilities at runtime (the Observers and the Ability Modeler).
Accountability	The toolkit provides a rich set of interface adaptations to implement based on user abilities detected (the UI Adapter).
Availability	The toolkit does not require specialized hardware, but instead is designed for easily accessible Android smartphones.
Adaptability	The toolkit enables real-time adaptations of mobile user interfaces, responsive to the user abilities detected at runtime.
Transparency	As future work, we could examine how to give users more control over developer-specified ability-based adaptations.
Performance	The toolkit provides a set of human performance metrics (the Observers) to model various aspects of user abilities and performances.
Context	The toolkit not only detects and understands inherent user abilities such as fine motor control, but also "situational" abilities under certain contexts, such as having tremor while walking or running.

Table 3. The seven principles of Ability-Based Design [62] and how ABD-MT addresses each principle.

5.3 Enabling Runtime Interface Adaptations

Distinct from prior work on "generating" an optimal user interface, our priority in this work was instead to provide a rich API of methods for developers to choose from. Our example apps do not *generate* interfaces for a user's abilities before the apps run (e.g., like SUPPLE [15, 16]), but instead dynamically *respond to* a user's observed abilities at runtime. Developers can thus control UI adaptations based on their target users and usage scenarios, in contrast to adaptations specified by end users themselves [43, 66]. Our example applications and participants' brainstorming ideas also do not aim for an "ideal" set of adaptations for each use case, but only serve as demonstrations of the capabilities and potentials of our toolkit. Additionally, to improve usability of these adaptations and prevent potentially frequent, undesired UI changes during runtime, our toolkit provides a variety of method overloads (Section 3.1.2, Section 3.2, Section 3.3.5) to inform adaptations based on observed interactions during a time window and to prevent the UI from constantly changing. That said, giving users control over how the ABD-MT customizes their UIs at runtime is an important direction for future work.

5.4 Extensibility and Toolkit Contributions

Although participants already found our toolkit applicable to a variety of use cases (as in Section 4.4.3 and Section 5.2), we do not claim ABD-MT to be comprehensive in addressing all possible abilities and adaptations. For example, additional abilities to detect can include lighting conditions, noise levels, gaze location, posture, and grip. That said, our toolkit provides a powerful abstraction through its three-part architecture which allows addition of other Observers with associated methods in the Ability Modeler and the UI Adapter. Aside from the usability of our toolkit and the variety of capabilities it already has, the core contribution of this work is its architectural separation.

Another possible limitation in the current toolkit design is the way it addressed the combination of abilities. While explicitly exporting methods involving different *Ability Groups* (e.g. `hasWalkingTremor`) can potentially help increase developer awareness of the full toolkit features, it can also seem rigid or difficult to expand to more abilities. Developers can of course always author adaptations based on arbitrary combinations of abilities, but our examination of participant

ideated apps (Section 4.4.4) found none of the apps were associated with more than two Ability Groups, making our method naming choice suitable for use in practice. That said, future work might explore higher levels of abstractions to support easier combination of multiple abilities.

As the goal of ABD-MT is to support development of ability-based apps, we also examined our toolkit relative to the seven principles of Ability-Based Design (Table 3): Ability, Accountability, Availability, Adaptability, Transparency, Performance, and Context [62]. Although the toolkit addresses a majority of these principles, we also do not claim ABD-MT to be comprehensive in addressing all of them. For example, the previously-noted opportunity for future work providing additional user control over developer-specified adaptations (Section 5.3) would improve Transparency.

6 Limitations and Future Work

As with all other toolkits, our toolkit has its limitations in not comprehensively addressing every aspect of user ability with every possible adaptation. As noted in Section 5.4, there are a variety of abilities and situations that our toolkit can be potentially extended to in future work, such as gait [17], hand posture and grip [19], accurate gaze location [27, 37, 56], temperature [48], noise [47], lighting [3], and more, by adding new *Observers* and new methods to the *Ability Modeler*.

In this toolkit, we favored relatively straightforward touch and gesture sensing and modeling techniques [26, 28, 29, 36], rather than complex machine learning models [38, 40, 58]. We also relied upon the built-in Android activity recognition and vision sensing APIs for physical activities [32] and visual attention [33], without pursuing accurate gaze location through more sophisticated eye-tracking techniques [31, 37, 42, 56]. Extending the *UI Adapter* with additional capabilities and user control over UI adaptations is also an important direction for our future work.

In our evaluation, we compared the lines of code with and without the toolkit for our five example applications. Our determination of the lines-of-code reduction is based on our understanding of the best implementation of each app, thus it serves as a baseline for demonstrating code reduction, but could vary with different implementations. We did not ask each study participant to implement each design with and without ABD-MT as it would have required hours of additional time from each participant. Additionally, our participants in the empirical study were all college students and mostly novices, which demonstrated a relatively low learning barrier of the toolkit, but is also not representative of the broader population of developers. Furthermore, we focused on evaluating the usability of the toolkit itself in supporting development of ability-aware apps, while the end-user usability of developer-created apps was not evaluated. For future work, we might run field experiments to compare the lines of code and the amount of development time required with and without the toolkit on a larger set of applications, conduct additional studies with end users to understand the usability of real-time adaptations in apps created by developers, and deploy the toolkit with more developers with varying backgrounds to better understand how developers might use our toolkit.

Additionally, as mentioned in Section 5.3, while this work focuses on enabling a rich set of adaptations for developers to choose based on their target users and scenarios, giving users control over how ABD-MT customizes their UIs at runtime is also an important avenue for future work.

7 Conclusion

We have presented the Ability-Based Design Mobile Toolkit (ABD-MT), a mobile toolkit that enables the creation of apps capable of detecting and responding to users' situated abilities at runtime. The ABD-MT architecture consists of three core components: the *Observers* capture user behavior and sensor data, exposing methods for human performance metrics; the *Ability Modeler* consumes data from the *Observers* and forms a higher-level, holistic understanding of a user's situated abilities;

and finally, the *UI Adapter* provides methods for manipulating UI widgets for runtime adaptations. We demonstrated the capabilities of ABD-MT through five example apps that showcased how a variety of adaptations in response to various user scenarios. Our example applications showed how ABD-MT simplifies the development of ability-aware apps by achieving ability-based functionality in 91.5% fewer lines of code. We also demonstrated how ABD-MT simplifies development of ability-aware apps with a study with 11 programmers who had Android development experience, showing that our toolkit is easy to learn, use, and apply to a variety of end-user scenarios. It is our hope that ABD-MT will lead to additional research in toolkit support for more accessible mobile apps and devices capable of better matching and responding to their users' situated abilities.

Acknowledgments

This work was supported in part by Google, by Toyota Research Institute, by the University of Washington Center for Research and Education on Accessible Technology and Experiences (CREATE), and by National Science Foundation award #IIS-1702751. Any opinions, findings, conclusions or recommendations expressed in our work are those of the authors and do not necessarily reflect those of any supporter.

References

[1] Ahmad Alsaleem, Ross Imburgia, Andrew Merryweather, Jeffrey Rosenbluth, Stephen Trapp, and Jason Wiese. 2020. Applying Ability-Based Design Principles to Adaptive Outdoor Activities. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*. 1–12.

[2] Apple. 2022. Turn Attention Aware features on or off on your iPhone or iPad Pro. https://support.apple.com/en-us/HT208245

[3] Leon Barnard, Ji Soo Yi, Julie A Jacko, and Andrew Sears. 2007. Capturing the effects of context on human performance in mobile computing systems. *Personal and Ubiquitous Computing* 11, 2 (2007), 81–96.

[4] Andrew A Bayor. 2019. HowToApp: Supporting life skills development of young adults with intellectual disability. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*. 697–699.

[5] Xiaojun Bi, Yang Li, and Shumin Zhai. 2013. FFitts law: modeling finger touch with fitts' law. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 1363–1372.

[6] Xiaojun Bi and Shumin Zhai. 2013. Bayesian touch: a statistical criterion of target selection with finger touch. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*. 51–60.

[7] Huw W Bristow, Chris Baber, James Cross, James F Knight, and Sandra I Woolley. 2004. Defining and evaluating context for wearable computing. *International Journal of Human-Computer Studies* 60, 5-6 (2004), 798–819.

[8] Anind K Dey, Gregory D Abowd, and Daniel Salber. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human–Computer Interaction* 16, 2-4 (2001), 97–166.

[9] Tilman Dingler, Albrecht Schmidt, and Tonja Machulla. 2017. Building cognition-aware systems: A mobile toolkit for extracting time-of-day fluctuations of cognitive performance. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 1–15.

[10] Paul Dourish. 2004. What we talk about when we talk about context. *Personal and ubiquitous computing* 8, 1 (2004), 19–30.

[11] Rachel L Franz, Jacob O Wobbrock, Yi Cheng, and Leah Findlater. 2019. Perception and adoption of mobile accessibility features by older adults experiencing ability changes. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*. 267–278.

[12] Krzysztof Z Gajos, Amy Hurst, and Leah Findlater. 2012. Personalized dynamic accessibility. *Interactions* 19, 2 (2012), 69–73.

[13] Krzysztof Z Gajos, Jing Jing Long, and Daniel S Weld. 2006. Automatically generating custom user interfaces for users with physical disabilities. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*. 243–244.

[14] Krzysztof Z Gajos, Daniel S Weld, and Jacob O Wobbrock. 2010. Automatically generating personalized user interfaces with Supple. *Artificial intelligence* 174, 12-13 (2010), 910–950.

[15] Krzysztof Z Gajos, Jacob O Wobbrock, and Daniel S Weld. 2007. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. 231–240.

[16] Krzysztof Z Gajos, Jacob O Wobbrock, and Daniel S Weld. 2008. Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In *Proceedings of the ACM SIGCHI conference on Human Factors in Computing Systems*. 1257–1266.

[17] Mayank Goel, Leah Findlater, and Jacob Wobbrock. 2012. WalkType: using accelerometer data to accomodate situational impairments in mobile touch screen text entry. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 2687–2696.

[18] Mayank Goel, Alex Jansen, Travis Mandel, Shwetak N Patel, and Jacob O Wobbrock. 2013. ContextType: using hand posture information to improve mobile touch screen text entry. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 2795–2798.

[19] Mayank Goel, Jacob Wobbrock, and Shwetak Patel. 2012. Gripsense: using built-in sensors to detect hand posture and pressure on commodity mobile phones. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. 545–554.

[20] Simon Harper. 2007. Is there design-for-all? *Universal Access in the Information Society* 6, 1 (2007), 111–113.

[21] Christian Holz and Patrick Baudisch. 2010. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 581–590.

[22] Christian Holz and Patrick Baudisch. 2011. Understanding touch. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 2501–2510.

[23] Meike Jipp, Essam Badreddin, Ciamak Abkai, and Jurgen Hesser. 2008. Individual ability-based system configuration cognitive profiling with bayesian networks. In *2008 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 3359–3364.

[24] Meike Jipp, Christian Bartolein, and Essameddin Badreddin. 2009. Predictive validity of wheelchair driving behavior for fine motor abilities: Definition of input variables for an adaptive wheelchair system. In *2009 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 39–44.

[25] Meike Jipp, Christian Bartolein, Essam Badreddin, Ciamak Abkai, and Jürgen Hesser. 2009. Psychomotor profiling with Bayesian neworks. In *2009 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 1680–1685.

[26] Simeon Keates, Faustina Hwang, Patrick Langdon, P John Clarkson, and Peter Robinson. 2002. Cursor measures for motion-impaired computer users. In *Proceedings of the Fifth International ACM Conference on Assistive Technologies*. 135–142.

[27] Andy Kong, Karan Ahuja, Mayank Goel, and Chris Harrison. 2021. EyeMU Interactions: Gaze+ IMU Gestures on Mobile Devices. In *Proceedings of the 2021 International Conference on Multimodal Interaction*. 577–585.

[28] Junhan Kong, Mingyuan Zhong, James Fogarty, and Jacob O Wobbrock. 2021. New metrics for understanding touch by people with and without limited fine motor function. In *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*. 1–4.

[29] Junhan Kong, Mingyuan Zhong, James Fogarty, and Jacob O Wobbrock. 2022. Quantifying Touch: New Metrics for Characterizing What Happens During a Touch. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–13.

[30] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation strategies for HCI toolkit research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–17.

[31] Dongheng Li, David Winfield, and Derrick J Parkhurst. 2005. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops*. IEEE, 79–79.

[32] Google LLC. 2021. Android Activity Recognition Transition API, Google Play Services. https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionClient

[33] Google LLC. 2021. Google Vision API, Google Play Services. https://developers.google.com/android/reference/com/google/android/gms/vision/face/package-summary/

[34] Elan D Louis. 2019. Tremor. *CONTINUUM: Lifelong Learning in Neurology* 25, 4 (2019), 959–975.

[35] Miroslav Macik, Tomas Cerny, and Pavel Slavik. 2014. Context-sensitive, cross-platform user interface generation. *Journal on Multimodal User Interfaces* 8, 2 (2014), 217–229.

[36] I Scott MacKenzie, Tatu Kauppinen, and Miika Silfverberg. 2001. Accuracy measures for evaluating computer pointing devices. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 9–16.

[37] Alexander Mariakakis, Mayank Goel, Md Tanvir Islam Aumi, Shwetak N Patel, and Jacob O Wobbrock. 2015. SwitchBack: Using focus and saccade tracking to guide users' attention for mobile task resumption. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 2953–2962.

[38] Alex Mariakakis, Sayna Parsi, Shwetak N Patel, and Jacob O Wobbrock. 2018. Drunk user interfaces: Determining blood alcohol level through everyday smartphone tasks. In *Proceedings of the ACM SIGCHI Conference on Human*

Factors in Computing Systems. 1–13.

[39] Sachi Mizobuchi, Mark Chignell, and David Newton. 2005. Mobile text entry: relationship between walking speed and text input task difficulty. In *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*. 122–128.

[40] Martez E Mott and Jacob O Wobbrock. 2019. Cluster Touch: Improving touch accuracy on smartphones for people with motor and situational impairments. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 1–14.

[41] Maia Naftali and Leah Findlater. 2014. Accessibility in context: understanding the truly mobile experience of smartphone users with motor impairments. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility*. 209–216.

[42] Takehiko Ohno, Naoki Mukawa, and Atsushi Yoshikawa. 2002. FreeGaze: a gaze tracking system for everyday gaze interaction. In *Proceedings of the 2002 Symposium on Eye Tracking Research & Applications*. 125–132.

[43] Yi-Hao Peng, Muh-Tarng Lin, Yi Chen, TzuChuan Chen, Pin Sung Ku, Paul Taele, Chin Guan Lim, and Mike Y Chen. 2019. PersonalTouch: Improving touchscreen usability by personalizing accessibility settings based on individual user's touchscreen interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–11.

[44] Sayan Sarcar, Jussi Jokinen, Antti Oulasvirta, Xiangshi Ren, Chaklam Silpasuwanchai, and Zhenxin Wang. 2017. Ability-based optimization: Designing smartphone text entry interface for older adults. In *IFIP Conference on Human-Computer Interaction*. Springer, 326–331.

[45] Sayan Sarcar, Jussi PP Jokinen, Antti Oulasvirta, Zhenxin Wang, Chaklam Silpasuwanchai, and Xiangshi Ren. 2018. Ability-based optimization of touchscreen interactions. *IEEE Pervasive Computing* 17, 1 (2018), 15–26.

[46] Sayan Sarcar, Jussi Joklnen, Antti Oulasvirta, Chaklam Silpasuwanchai, Zhenxin Wang, and Xiangshi Ren. 2016. Towards ability-based optimization for aging users. In *Proceedings of the International Symposium on Interactive Technology and Ageing Populations*. 77–86.

[47] Zhanna Sarsenbayeva, Niels van Berkel, Eduardo Velloso, Vassilis Kostakos, and Jorge Goncalves. 2018. Effect of distinct ambient noise types on mobile interaction. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 2 (2018), 1–23.

[48] Zhanna Sarsenbayeva, Niels Van Berkel, Aku Visuri, Sirkka Rissanen, Hannu Rintamaki, Vassilis Kostakos, and Jorge Goncalves. 2017. Sensing cold-induced situational impairments in mobile interaction using battery temperature. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 1–9.

[49] Bill Schilit, Norman Adams, and Roy Want. 1994. Context-aware computing applications. In *1994 first workshop on mobile computing systems and applications*. IEEE, 85–90.

[50] William Noah Schilit. 1995. *A system architecture for context-aware mobile computing*. Columbia University.

[51] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. 1999. Advanced interaction in context. In *International Symposium on Handheld and Ubiquitous Computing*. Springer, 89–101.

[52] Albrecht Schmidt, Michael Beigl, and Hans-W Gellersen. 1999. There is more to context than location. *Computers & Graphics* 23, 6 (1999), 893–901.

[53] Andrew Sears, Min Lin, Julie Jacko, and Yan Xiao. 2003. When computers fade: Pervasive computing and situationally-induced impairments and disabilities. In *HCI international*, Vol. 2. 1298–1302.

[54] Andrew Sears, Mark Young, and Jinjuan Feng. 2009. Physical disabilities and computing technologies: an analysis of impairments. In *Human-Computer Interaction*. CRC Press, 87–110.

[55] Daniil Umanski and Yael Avni. 2017. PLAY-ABLE: developing ability-based play activities for children with special needs. In *Proceedings of the 11th International Convention on Rehabilitation Engineering and Assistive Technology*. 1–4.

[56] Nachiappan Valliappan, Na Dai, Ethan Steinberg, Junfeng He, Kantwon Rogers, Venky Ramachandran, Pingmei Xu, Mina Shojaeizadeh, Li Guo, Kai Kohlhoff, et al. 2020. Accelerating eye movement research via accurate and affordable smartphone eye tracking. *Nature communications* 11, 1 (2020), 4553.

[57] Daniel Vogel and Patrick Baudisch. 2007. Shift: a technique for operating pen-based interfaces using touch. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 657–666.

[58] Daryl Weir, Simon Rogers, Roderick Murray-Smith, and Markus Löchtefeld. 2012. A user-specific machine learning approach for improving touch accuracy on mobile devices. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. 465–476.

[59] Paul Whittington, Huseyin Dogan, Keith Phalp, and Nan Jiang. 2020. Detecting physical abilities through smartphone sensors: an assistive technology application. *Disability and Rehabilitation: Assistive Technology* (2020), 1–12.

[60] Daniel Wigdor, Clifton Forlines, Patrick Baudisch, John Barnwell, and Chia Shen. 2007. Lucid touch: a see-through mobile device. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. 269–278.

[61] Jacob O Wobbrock. 2019. Situationally-induced impairments and disabilities. In *Web accessibility*. Springer, 59–92.

[62] Jacob O Wobbrock, Krzysztof Z Gajos, Shaun K Kane, and Gregg C Vanderheiden. 2018. Ability-based design. *Commun. ACM* 61, 6 (2018), 62–71.
[63] Jacob O Wobbrock, Shaun K Kane, Krzysztof Z Gajos, Susumu Harada, and Jon Froehlich. 2011. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing (TACCESS)* 3, 3 (2011), 1–27.
[64] Jacob O Wobbrock, Brad A Myers, and John A Kembel. 2003. EdgeWrite: a stylus-based text entry method designed for high accuracy and stability of motion. In *Proceedings of the 16th Annual ACM symposium on User Interface Software and Technology*. 61–70.
[65] Flynn Wolf, Ravi Kuber, Dianne Pawluk, and Brian Turnage. 2017. Towards supporting individuals with situational impairments in inhospitable environments. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. 349–350.
[66] Jason Wu, Gabriel Reyes, Sam C White, Xiaoyi Zhang, and Jeffrey P Bigham. 2021. When can accessibility help? an exploration of accessibility feature recommendation on mobile devices. In *Proceedings of the 18th International Web for All Conference*. 1–12.
[67] Yeliz Yesilada and Simon Harper. 2019. *Web Accessibility: A Foundation for Research*. Springer.

A The ABD-MT Pseudocode

In Figure 9, we provide the pseudocode showing how the ABD-MT observes user behavior related to touch, gesture, physical activity, and attention.

B Full List of Methods Currently Supported by the ABD-MT

In Figure 10, we provide a full list of API methods currently supported by the ABD-MT, including methods from the *Observers*, the *Ability Modeler*, and the *UI Adapter*, each in a separate column. The documentation of the ABD-MT is also included in the supplementary materials and available at https://github.com/abdmt-toolkit/abdmt/wiki. It is expected that over time, the API will expand in scope and capability.

C Participant Brainstorming Ideas

In Table 4, we provide a full list of participants' brainstorming ideas of apps that can be potentially built with the ABD-MT.

Received February 2024; revised May 2024; accepted June 2024

```
Observing Touch and Gesture
```

```
TOUCH-EVENT-DISPATCHER(event):
 1 ADD(event)
 2 GESTURE-DETECTOR(event)
 3 if currentPath not empty and event is TOUCH_UP:
 4     RECORD-TOUCH()
GESTURE-DETECTOR(event):
 1 onSingleTap(): RECORD-TOUCH()
 2 onFling(): RECORD-GESTURE()
ADD(event):
 1 if event is TOUCH_DOWN:
 2   currentPath←empty
 3 get x,y,major,minor,area,time from event
 4 pt←TimePoint(x,y,major,minor,area,time)
 5 append pt to currentPath
RECORD-TOUCH():
 1 touchObserver.add(currentPath)
 2 currentPath←empty
RECORD-GESTURE():
 1 gestureObserver.add(currentPath)
 2 currentPath←empty
```

```
Observing Physical Activity
```

```
ON-ACTIVITY-TRANSITION-DETECTED(
  activityTansitionEvents):
 1 foreach event in activityTransitionEvents:
 2   get activityType,transitionType,time
 3   case TransitionType of:
 4     ENTER:
 5       currentActivity←Activity(activityType,time)
 6     EXIT:
 7       currentActivity.endTime←time
 8       activityObserver.add(currentActivity)
```

```
Observing Attention
```

```
ON-EYES-APPEAR-IN-VIEW():
 1 if currentAttention not empty:
 2   currentAttention.endTime←time
 3   attentionObserver.add(currentAttention)
 4 currentAttention←Attention(LOOKING,time)
ON-EYES-DISAPPEAR-IN-VIEW():
 1 if currentAttention not empty:
 2   currentAttention.endTime←time
 3   attentionObserver.add(currentAttention)
 4 currentAttention←Attention(NOT_LOOKING,time)
```

Fig. 9. Pseudocode showing how the ABD-MT observes touch, gesture, physical activity, and attention behaviors. Relevant events are observed by our toolkit *before* being dispatched in the usual manner to the client application.

Fig. 10. Full list of methods supported by the ABD-MT.

	P#	App Description	App Type
1	P3	A banking app that automatically hides screen content when the user looks away, especially when they are typing their password or transferring money	Banking
2	P3	A browser that automatically enlarges buttons when a user misclick too many times	Browser
3	P2	A calendar app that enlarges widgets and displays your destination based on your calendar events when you are walking	Calendar
4	P8	A calendar app that automatically displays the next events when a user is walking between buildings	Calendar
5	P2	A drawing app that enlarges control widgets and smoothes the drawing line if the user has tremor	Creativity
6	P10	A drawing app that automatically smoothes the lines if a user is detected having tremor	Creativity
7	P8	A developer testing app that dynamically resizes the widgets for developers with tremor or changes the screen contrast for developers low vision, and changes the workflow of going through the testing steps	Development Tool
8	P10	An email app that enlarges buttons if a user has tremor	Emails
9	P7	A farm monitoring app (of counting number of animals) that enlarges the counter and existing markers when a user gets distracted and looks back at the app	Farm Management
10	P5	A food ordering app that displays the top 5 restaurants in the user's favorites when the user is driving	Food Ordering
11	P3	A mobile game (of catching eggs on the screen) that automatically pauses and displays a pausing screen when the user looks away	Game
12	P3	A mobile game (of catching eggs on the screen) that pops up an alert when the user is driving or cycling while using gestures	Game
13	P9	A mobile game that enlarges buttons based on a user's touch patterns	Game
14	P2	Only display a summary of all notifications when the user is walking or running	General
15	P6	Displays an alert for the user to stop walking when they are looking at the phone while walking	General
16	P6	Squishes app display to one side of the screen if the user is using one hand	General
17	P6	Displays an alert for the user to confirm they are clicking on the intended target if the user has tremor	General
18	P6	Displays an alert that reminds a user not to use the phone when the user is driving while still having touches on the screen	General
19	P6	Displays larger text and buttons for older adults who have tremor; hide widgets they don't need to use	General
20	P7	Hides everything except the time and map to avoid distractions when a user is walking or driving	General
21	P7	Enlarges the screen widget a user is looking at when they are running	General
22	P9	Enlarges all text fonts if a user brings the phone too close to their face	General
23	P9	Enlarges all fonts and widgets for users with tremor	General
24	P9	Hides everything except map and emergency call when a user is driving	General
25	P3	A fitness app that asks the user "are you currently running" and turns the running timer on automatically when the user is detected running	Health Tracking
26	P5	A health monitoring app that evaluates tremor level of users with Parkinson's disease and send the information to their doctors	Health Tracking
27	P4	A piano tutorial app that adapts the key playback time according to how long a user is pressing the buttons	Learning
28	P1	A map app that tells the user where they should go when the user is running and looking at the screen	Map
29	P1	A map app that displays a list of the nearby restaurants when the user is walking, and shows the menu of the closest restaurant when the user stops	Map
30	P1	A map app that enlarges buttons and displays a larger keyboard when the user is walking or running so the user can type in the address more easily	Map
31	P1	A map app that enlarges the direction texts when the user is looking at the screen	Map
32	P6	A map app that enlarges all butons if the user is running	Map
33	P6	A map app that automatically displays the route to destination when the user is cycling, and enlarges buttons on the screen	Map
34	P8	A map app that displays the default view when a user is sitting or standing still, and automatically switches to the driving display with enlarged content when the user is detected driving	Map
35	P11	A map app that zooms in and enlarges widgets for users with tremor	Map
36	P1	A messaging app that displays some shortcuts for typing things like "on my way" when the user is walking / running / driving	Messaging
37	P5	A messaging app that displays a few shortcut messages like "I'm driving" / "be there in 5", "[current location]" and "send to partner" when the user is driving	Messaging
38	P6	A messaging app that enlarges keyboard sizes when the user is walking, running, or cycling	Messaging
39	P7	A messaging app that enlarges widgets and increases height of keyboard when a user is on a bus and has tremor	Messaging
40	P8	A messaging app that only displays the most recent historical messages when a user is walking or driving	Messaging
41	P11	A messaging app that disables interactive widgets if a user tries to walk and text at the same time	Messaging
42	P1	A music player app that increases the volume when the user is running	Music Player
43	P5	A music player that displays only a few buttons when the user is walking or running	Music Player
44	P8	A music app that displays the default view when a user is sitting or standing still, and automatically switches to the driving display when the user is detected driving	Music Player
45	P10	A music player that only allows a users to quickly glance over but not be able to interact with it while driving	Music Player
46	P10	A music player that applies a smoothing for scrolling selection of songs for a user with tremor	Music Player
47	P10	A notebook app that enlarges widgets and fonts if a user has difficulty reading and brings the phone too close to their face	Notebook
48	P3	A phone call app that automatically picks up the phone when the user looks at the screen while they are driving	Phone Call
49	P1	A focus monitoring app that locks the phone for some time if the user is currently in Work Mode but registers too many taps or gestures on the screen	Productivity Monitoring
50	P4	A monitoring app that computes how much time a user spends on social media and alerts the user when they read a limit (accurately track only when the user is looking at the screen)	Productivity Monitoring
51	P1	A reading or news app that enlarges fonts on the screen when the user is looking at the phone without wearing their glasses as usual	Reading
52	P5	A reading app that displays an alert to remind users with ADHD to focus on their reading when the user is looking away	Reading
53	P6	A reading or news app that enlarges text fonts when the environment is dark	Reading
54	P8	A reading app that displays different font sizes and enables different actions for navigating between text when a user is detected walking or on a vehicle	Reading
55	P8	A reading app that automatically switches to audio-only mode when a user is driving, and reverts back when they stop	Reading
56	P3	A ridesharing app that calls 911 or send a message to your emergency contact when you are supposed to be on a vehicle but are not	Safety Monitoring
57	P8	A safety monitoring app that sends a message to a user's emergency contact if the user is heading towards a location at late night, but is in a vehicle when they are supposed to be walking	Safety Monitoring
58	P1	A social media app (e.g. Twitter) that displays an ad only when the user is no looking at the screen for users' sake; the same app that displays an ad only when the user is looking at the screen for the company's sake	Social Media
59	P3	A social media app (e.g. Instagram) extension that allows users to choose to hide some widgets if they don't like those widgets (e.g. Instagram reels)	Social Media
60	P4	A social media app that automatically resizes texts and widgets based on the user's touch patterns	Social Media
61	P9	A short video app that only plays videos of certain types or with certain tags when a user is detected at home	Social Media
62	P11	A social media app that enlarges like or comment buttons based on a user's touch patterns	Social Media
63	P3	A spreadsheet app that enlarges the cell the user is currently working on, especially when the user needs to enter long text	Spreadsheets
64	P11	A video conferencing app that reminds the speaker that the participants are not looking at their screens	Video Conferencing
65	P2	A video player app that automatically turns up volume if the noise level is high	Video Player
66	P7	A video player that pauses and deactivates all the video playback control widgets when a user is walking and not looking at the screen	Video Player
67	P7	A video player that minimizes when a user is driving to avoid distraction	Video Player
68	P8	A video player that automatically switches to podcast mode and allows a user to interact in a less visual way when the user starts walking, running, or driving	Video Player
69	P10	A video player app that resizes dynamically according to a user's touch patterns as the playback control can be too small	Video Player

Table 4. Apps Ideated by Participants.